Lenguajes de Transformación de Modelos. Un análisis comparativo.

Natalia Correa
LIFIA, Facultad de Informática, Universidad de La Plata
La Plata, Argentina
natalia.correa@lifia.info.unlp.edu.ar

and

Roxana Giandini
LIFIA, Facultad de Informática, Universidad de La Plata
La Plata, Argentina
roxana.giandini@lifia.info.unlp.edu.ar

Abstract

The use of models and transformations among them are the primary concepts of OMG's proposal: MDA (Model Driven Architecture). Specifying transformation has become one of the most important tasks for the proposal, as evidenced by the development of several transformations languages. Studying these languages represents a useful tool for the computing science community whose desire is getting involved with MDA. The goal of this paper is to present a taxonomic framework for analyzing model transformation languages and then to make a comparison among some proposed languages (its features) by using such taxonomy. It is expected that this study reports on benefits to the state of the art of model transformation languages.

Keywords: Software Engineering, Model Driven Architecture, Model Transformation.

Resumen

La utilización de modelos y las transformaciones entre los mismos son conceptos primarios en la propuesta MDA (Arquitectura Dirigida por Modelos) de la OMG. Especificar transformaciones se ha vuelto una de las tareas más importantes, lo cual se ve reflejado en la creación de muchos lenguajes de transformación de modelos. Estudiar y conocer estos lenguajes resulta sumamente útil para toda la comunidad informática que desee comenzar a involucrarse y conocer más sobre esta propuesta. El objetivo de nuestro artículo es presentar un esquema taxonómico para analizar lenguajes de transformación de modelos y evaluar comparativamente algunos de ellos (sus características) en base a esta taxonomía. Esperamos que el análisis realizado constituya un aporte al estado del arte de los lenguajes actualmente definidos.

Palabras claves: Ingeniería de Software, Arquitectura Dirigida por Modelos, Transformación de Modelos.

1 INTRODUCTION

La propuesta MDA (Model Driven Architecture- Arquitectura Dirigida por Modelos)[1] de la OMG (Object Management Group)[2] presenta un proceso de desarrollo de software concebido para dar

soporte al desarrollo de sistemas, donde los conceptos más importantes son los modelos y las transformaciones entre ellos que generan a su vez, otros modelos. Estos se convierten, entonces, en los guías del desarrollo de software. En primer lugar, y según es propuesto por MDA, se definen uno o más modelos PIM (Platform-Independent Model) en algún lenguaje específico y que son independientes de cualquier plataforma de desarrollo. Estos PIMs se traducen en uno o más modelos PSM (Platform-Specific Model) que son específicos de la plataforma donde se ejecuten.
Esta "traducción" entre PIMs y PSMs se conoce como "transformación de modelos".
Describir transformaciones de modelos requiere de lenguajes específicos para la definición de las mismas. Actualmente, existen varias propuestas de lenguajes, muchas de ellas basadas en el estándar de la OMG, QVT (Query/View/Transformation) [3].
Entre los lenguajes definidos existen gran variedad de "tipos": icónicos y textuales, declarativos, operacionales y declarativos-operacionales, algunos basados en QVT y no otros no, compatibles con MOF[4], aquellos que soportan OCL[5], los que proveen traceability, los que proveen composición de transformaciones y hasta aquellos a los que se les ha implementado una herramienta CASE o un plugin para Eclipse.
Estudiar y conocer los lenguajes de transformación que sirven de soporte a la propuesta MDA es útil a toda la comunidad de informáticos. El objetivo de este artículo es, en base al estudio de los lenguajes de transformación, proponer una extensión a una clasificación de lenguajes existente y analizar algunos de los lenguajes que más crecimiento han tenido y que son los más utilizados y nombrados en la comunidad MDA, tomando a esta clasificación como referencia.
La organización del artículo es la siguiente: en la sección 2 presentamos algunos de los lenguajes existentes e introducimos una clasificación para el estudio de los lenguajes de transformación. En la sección 3, realizamos una comparación de lenguajes completando la taxonomía definida en la sección 2 con una selección de lenguajes de transformación de modelos definidos; para finalizar, en la sección 4 presentamos las conclusiones y líneas de trabajo futuro.

2 ESQUEMA DE CLASIFICACIÓN PARA LOS LENGUAJES DE TRANSFORMACIÓN

Desde la aparición de la metodología MDA, mucho se ha propuesto y definido en cuanto a lenguajes y herramientas que sirven de soporte y automatizan sus diferentes aspectos. Uno de estos aspectos donde se ha puesto más énfasis es en la definición de lenguajes que permiten traducir un modelo en otro, pasando desde PIMs a PSMs según indica MDA.

Antes de continuar, enunciamos algunas definiciones de conceptos de lenguajes de transformación. En la jerga MDA, una *transformación* es la generación automática de un modelo de salida o target a partir de un modelo de entrada o source y de acuerdo a la *descripción de una transformación*. Esta descripción se compone de una o más *reglas de transformación* que describen cómo un modelo source puede transformarse en un modelo target (cada uno en sus respectivos lenguajes). Finalmente, una *regla de transformación* describe cómo un elemento del source puede ser transformado en uno o más elementos del target.

2.1 Lenguajes de Transformación de Modelos. Propuestas Existentes

Nuestro trabajo comenzó con la investigación de los lenguajes existentes, muchos de los cuales son presentados a continuación. Como se verá, las propuestas actuales son muchas y muy diversas. Esto hace que no resulte una tarea sencilla la toma de decisión sobre qué lenguaje utilizar. La lista de

lenguajes incluye una breve referencia de cada uno de ellos. Para su posterior análisis, sólo fueron seleccionados algunos de los mencionados.

Tabla 1.Lenguajes de transformación de modelos. Una lista preliminar

Lenguaje	Características
ATL (Atlas Transformation Language) [6]	Lenguaje de transformación de modelos y herramienta en Eclipse desarrollada por el Atlas group (INRIA)
BOTL(Basic Object-Oriented Transformation Language) [7]	Propuesta gráfica de la Universidad Técnica de München
GreAT (Graph Rewriting and Transformations) [8]	Basado en la transformación de grafos. Es la propuesta de una organización independiente: ESCHER Research Institute (The Embedded Systems Consortium for Hybrid and Embedded Research)
JMI (Java Metadata Interface) [9]	Propuesta de Sun basado en MOF que permite manipulación de archivos XMI.
Kent o KMTL(Kent Model Transformation Language)[10]	Propuesta realizada por la Universidad de Kent
MTRANS [11]	Proyecto de la Universidad de Nantes. Es un framework que permite expresar transformaciones de modelos.
Mod-Transf [12]	Lenguaje de transformación de modelos y herramienta en Eclipse desarrollada por el Dart team (INRIA)
MOFScript [13]	Lenguaje de transformación modelo a texto (cuya propuesta pertenece a la OMG) y herramienta como plugin para Eclipse
MOLA (MOdel transformation LAnguage) [14]	Lenguaje gráfico para describir transformaciones propuesto por la Universidad de Letonia.
MT model transformation language [15]	Basado en QVT y desarrollado como DSL (Domain Specific Language) por L. Tratt del King's College de Londres.
MTL (Model Transformation Language) [16]	Lenguaje de transformación de modelos y herramienta en Eclipse desarrollada por el Triskell team (INRIA)
QVT (Query/View/ Transformation) [3]	Especificación estándar de OMG. Está basado en MOF (Meta Object Facility) para lenguajes de transformación en MDA.
Stratego [17]	Lenguaje de descripción de transformaciones de programas. La herramienta desarrollada como soporte del lenguaje es Stratego/XT
Tefkat [18]	Lenguaje declarativo basado en MOF y QVT. Es el aporte de la Universidad de Queensland.
UMLX [19]	Lenguaje gráfico que extiende a UML y también a QVT.
xUML (eXecutable UML) [20]	Propuesta basada en UML para la construcción de modelos de dominio ejecutables y sus transformaciones.

2.2 Clasificación Para La Evaluación de Lenguajes de Transformación

En la sección anterior se ha podido ver que las propuestas de lenguajes de transformación de modelos son muchas y muy diversas. Es claro que a la hora de describir una transformación

aparece un abanico de opciones de las cuales elegir. Y surge, como trabajo adicional, la elección de un lenguaje. Resulta necesario entonces conocer las características de estos lenguajes. Esta necesidad fue vislumbrada por Czarnecki y Helsen [21], quienes propusieron una clasificación para lenguajes de transformación de modelos.
En nuestra propuesta, tomamos como base el trabajo realizado por ellos, reordenando algunas características y adicionando otras.
En la figura 1 puede observarse en color rosa la propuesta original y en color celeste la extensión sugerida en este artículo.

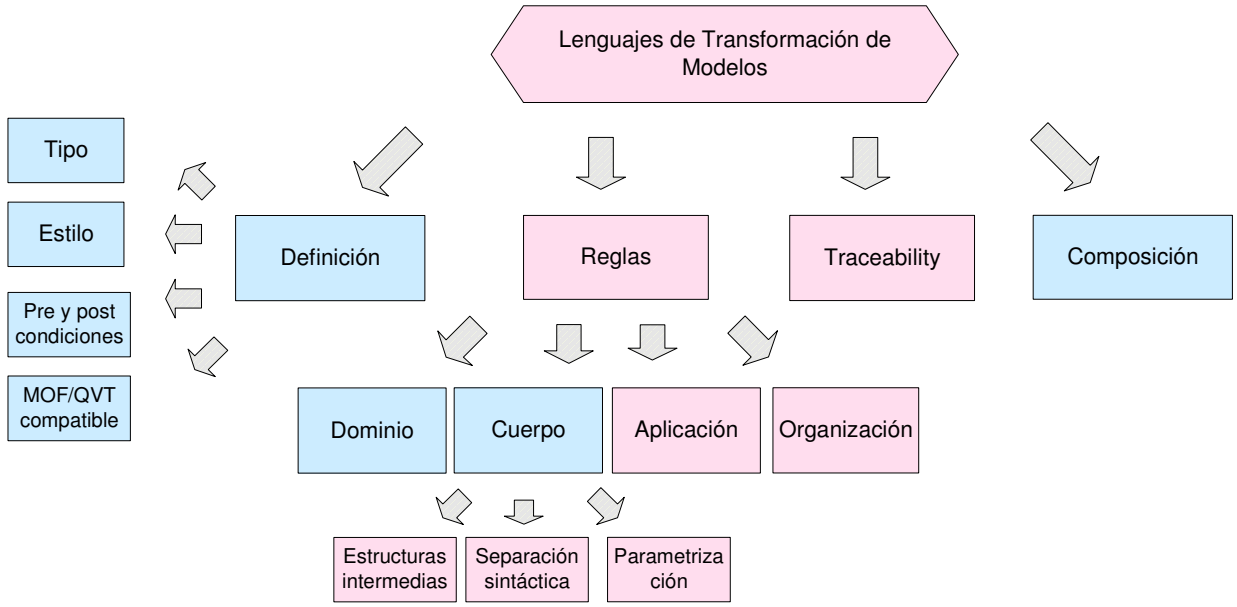

Figura 1. Clasificación propuesta

A continuación se presenta cada una de los aspectos representados en la figura 1, indicando a qué hace referencia cada uno.

Definición

La definición de una transformación determina características generales de la transformación. Se enuncian a continuación:

- **Tipo**: indica si la sintaxis del lenguaje es icónica (gráfica) o textual y si es modelo a modelo (M2M) o modelo a texto (M2Text).
- **Estilo:** un lenguaje puede ser declarativo, operacional o ambos
- **Pre y post condiciones:** si es posible especificarlos en la transformación y de qué forma puede hacerse (coloquial, con OCL, otro lenguaje). Lo deseable es que se puedan especificar y en lo posible, en OCL ya que es un lenguaje formal y estándar.
- **MOF/QVT compatible:** indica si el lenguaje se basa en los estándares definidos por OMG: si los metamodelos (lenguajes de dominio y codominio) que participan en la transformación son instancias de MOF y si el lenguaje de transformación se basa en QVT. Lo esperado es que los lenguajes sean compatibles con ambas propuestas.

Reglas de Transformación

Las reglas de transformación son unidades más pequeñas que componen una transformación. Como fue mencionado anteriormente, estas describen cómo un elemento del modelo de entrada puede ser transformado en uno o más elementos del modelo de salida.
Los aspectos a analizar son:

- **Dominio**: define uno o más modelos "de entrada" u origen o *source* y uno o más modelos "de salida" o destino o *target* sobre los cuales operarán las reglas de transformación
  - **Lenguajes del dominio**: cada dominio tiene un lenguaje asociado. Se espera que los lenguajes sean instancias de MOF.
  - **Dirección**: indica si los metamodelos source y target son *in*, *out*, o *in/out* (en el sentido en que se usan los parámetros en programación). Y si es unidireccional o bidireccional el sentido de la transformación
  - **Relación entre origen y destino de la transformación**: Si el source y el destino tienen el mismo o diferente modelo.

- **Cuerpo de la transformación**: en cuanto al cuerpo de la transformación, podemos decir que consta de:
  - Declaración de (meta)variables y sus tipos
  - Patrones de transformaciones
  - **Separación sintáctica**: algunos lenguajes separan claramente las partes de una regla de transformación que operan sobre el modelo source (LHS) de las partes que operan sobre el modelo target (RHS). La separación sintáctica hace a la legibilidad del lenguaje.
  - **Estructuras intermedias:** si el lenguaje recurre a alguna estructura adicional para describir la transformación y que no es parte del modelo a transformar.
  - **Parametrización**: el tipo más simple de parametrización es el uso de los parámetros del control que permiten el paso de valores.

- **Aplicación de las reglas**: como hemos mencionado, una regla se aplica a algún elemento del modelo de entrada. Como puede haber más de una regla que "machee" con un elemento particular, se debe definir alguna estrategia para determinar el orden de aplicación de las reglas.
  - **Orden:** la aplicación de las reglas puede ser determinístico o no determinístico.
  - **Aplicación condicional**: en algunas reglas de transformación se puede tener aplicación condicional de las mismas. En estos casos, existe una condición que debe ser verdadera para que la regla se ejecute.
  - **Iteración de reglas**: si el lenguaje provee estructuras de iteración, o mecanismos de recursión.

- **Organización de las reglas**: se refiere a la composición y estructuración de múltiples reglas. Se puede dividir en varios aspectos:
  - **Modularización:** en el caso en que se provea modularización de las reglas (agrupar un conjunto de reglas en un módulo, que puede ser llamado desde otros módulos).
  - **Mecanismos de reuso:** la definición de regla/s en base a otra/s.

Traceability

Hace referencia a aquellos mecanismos provistos por los lenguajes que permiten guardar ciertos aspectos de la ejecución de la transformación. Se pueden crear y mantener "conexiones" o *links* entre elementos del dominio y del codominio que mapean los dominios source y target, cada vez

que una regla de transformación es ejecutada.

Composición

La composición indica cómo pueden relacionarse un número de transformaciones para obtener una nueva. Pueden encadenarse dos o más transformaciones consecutivamente; pueden componerse dos o más transformaciones existentes en una nueva transformación con sus nuevas relaciones o pueden combinarse dos transformaciones de forma tal que se obtienen codominios más amplios.

3 COMPARACIÓN DE LOS LENGUAJES DE TRANFORMACIÓN

En base a los criterios presentados en la sección 2.2, analizamos algunos de los lenguajes presentados en la sección 2.1. Para tal análisis, se definieron dos (2) metamodelos y transformaciones entre los mismos (compuestas por varias reglas) y se utilizó cada uno de los lenguajes seleccionados para describir la transformación. Los ejemplos no son incluidos por razones de espacio.

La elección de estos lenguajes se hizo en base al crecimiento, aceptación y utilización que han tenido en la comunidad MDA. Los seleccionados para el análisis presentado fueron los siguientes: **ATL**, **MOFScript**, **MOLA**, **Tefkat** y **ULMX**. Cabe aclarar que todos los lenguajes elegidos cuentan con la implementación de una herramienta que da soporte a la definición del lenguaje. Excepto MOLA, los desarrollos de tales herramientas se han hecho bajo la plataforma y como plugins para Eclipse. Escapa a este trabajo, el análisis, descripción y clasificación de dichas herramientas.

3.1 Análisis de los Lenguajes Seleccionados

ATL (Atlas Transformation Language)

ATL es un lenguaje de transformación de modelos híbrido que permite, en su definición de transformaciones, especificar construcciones declarativas y operativas. La propuesta es del ATLAS Group del INRIA & LINA, de la Universidad de Nantes y fue desarrollado como parte de la plataforma AMMA (ATLAS Model Management Architecture).
ATL es un lenguaje modelo a modelo híbrido o mixto, en el sentido que permite construcciones tanto declarativas como imperativas.
Otros detalles de la Definición del lenguaje:

- Es compatible con los estándares de la OMG: es posible describir transformaciones modelo a modelo (y ambos deben ser instancias de MOF). Además, el lenguaje se basa en QVT.
- Permite definir pre y post condiciones en un lenguaje ya conocido: OCL

En cuanto a las reglas de transformación, ATL define un dominio (metamodelo) para el source y otro dominio para el target, siendo ambos instancias de MOF y con direcciones in y out respectivamente. Source y target pueden tener iguales o diferentes dominios (aunque sean iguales, ambos deben ser claramente identificados). Si bien las transformaciones son unidirecciones, ATL permite la definición de transformaciones bidireccionales como la implementación de dos transformaciones, una para cada dirección.

Como características particulares del lenguaje, podemos mencionar las estructuras que define este lenguaje. En primer lugar, la definición de transformaciones forman módulos (modules) que contienen las declaraciones iniciales y un número de *helpers* y reglas de transformación. Los helpers, son una estructura intermedia dentro de las transformaciones que facilitan la navegación, la modularización y el reuso. Permiten definir operaciones y tuplas OCL Existe también una construcción llamada *called rule,* y que representa a un procedure. Estas pueden contener argumentos y pueden ser invocadas por su nombre. Resulta sumamente expresivo y de fácil escritura para quienes conocen OCL (no es necesario aprender un nuevo lenguaje).

La aplicación de las reglas se realiza de forma no determinística, por "macheo" de reglas y no se ha provisto ninguna construcción o claúsula que permita aplicar en forma condicional las reglas. Cabe mencionar que la invocación de called rules es determinística. Esta invocación, junto con la utilización de parámetros permiten soportar recursión.

ATL provee modularización por sus procedimientos o called rules, además de que sus módulos pueden incluir a otros; y mecanismos de reuso ya que las reglas se pueden heredar.

En cuanto a la trazabilidad, este lenguaje crea un *traceability link* con la ejecución de cada regla que se guarda en el motor de la transformación. Este link relaciona a tres (3) elementos: la regla, el "macheo" (los elementos del source) y los elementos creados en el target.

ATL permite componer transformaciones mediante la definición de reglas.

Para finalizar, vale la pena mencionar que ATL se ha convertido en un lenguaje tan utilizado que ya se ha definido un repositorio de transformaciones entre diversos lenguajes (http://www.eclipse.org/m2m/atl/atlTransformations/) .Ya existen más de 60 trasformaciones documentadas y con los archivos fuente disponibles.

MOFScript

MOFScript es un lenguaje de transformación de modelo a texto presentado por la OMG. Este lenguaje presta particular atención a la manipulación de strings y de texto y al control e impresión de salida de archivos.
No sólo es un estándar, sino que además se basa en otros estándares de la OMG: es QVT compatible y MOF compatible con el modelo de entrada (el target siempre es texto).
Para las reglas de transformación, MOFScript define un metamodelo de entrada para el source y sobre el cual operarán las reglas. El target es generalmente un archivo de texto (o salida de texto por pantalla).
Las transformaciones son siempre unidireccionales y no es posible definir pre y post condiciones para ellas. La separación sintáctica resulta clara por la misma definición de reglas, lo que hace a la legibilidad del lenguaje. No provee estructuras intermedias, pero sí parametrización necesaria para la invocación de reglas.

En cuanto a la aplicación de reglas, podemos decir que se aplican en forma determinística y en orden secuencial. Se provee aplicación condicional e iteración de reglas. Las condiciones de aplicación se expresan en cláusulas *when*, como definición de guardas. La iteración se realiza mediante los iteradores *for each* y *while*.

MOFScript no organiza las reglas en módulos propiamente dichos. Ahora bien, en la definición de una regla se puede invocar a otras reglas, utilizando incluso parámetros, con lo cual se asemeja

bastante a un módulo. Y es posible definir jerarquías de transformaciones.

Finalmente, para *traceability*, MOFScript define (pero no implementa aún) un conjunto de conceptos para relacionar elementos del source con sus ubicaciones en los archivos de texto generados en el target.
En cuanto a la composición de transformaciones, no es algo que se haya pensado en este lenguaje aún.

MOLA (MOdel transformation LAnguage)

MOLA es un lenguaje gráfico de transformación de modelos, propuesto por la Universidad de Letonia. La intensión de este lenguaje es combinar la programación estructurada tradicional con reglas basadas en patrones de transformación.

La definición del lenguaje no se basó en QVT y los metamodelos que participan de la transformación pueden o no ser instancias de MOF; lo único restrictivo es que ambos deben ser definidos como dos modelos diferentes (source y target) aunque sean el mismo. El source es el modelo de entrada in/out (sus elementos pueden ser modificados) y el target de salida (sólo out). Las transformaciones son unidireccionales.
Si bien no se mencionan pre y post condiciones, el lenguaje provee notas donde pueden escribirse cláusulas OCL para la aplicación condicional de reglas. Estas podrían usarse entonces para pre y post condiciones.

En cuanto al cuerpo de la transformación, se ve que no hay separación sintáctica de los elementos que se transforman y de los que se crean. Tampoco provee estructuras intermedias.
El elemento principal del lenguaje es un concepto gráfico del *loop*, que se utiliza mucho para iterar sobre los elementos de los modelos que participan de la transformación. También para ello, permite definir variables locales a cada regla de transformación.

Sobre la aplicación de reglas, podemos decir que se aplican en forma determinística y en orden secuencial según son definidas. Existe la aplicación condicional mediante la adición de notas con cláusulas OCL. La iteración se realiza mediante dos tipos de loop: un loop de "tipo uno" que se ejecuta una vez para cada instancia válida del source (for each); un loop de "tipo dos" que continúa la ejecución mientras haya al menos una instancia válida en el source (while). El lenguaje utiliza patrones definidos para ser aplicados en las transformaciones.
Si bien gráficamente resulta intuitivo, las iteraciones pueden volverse en poco confusas, sobre todo si hay varias reglas anidadas.

En cuanto a *traceability*, MOLA permite definir "*mapping associations*" para tracear instancias entre modelos. Estas se definen en las reglas como notas anexadas entre los elementos del dominio y codominio que participan de la transformación.

La composición de transformaciones no ha sido tenida en cuenta en la definición del lenguaje.

Tefkat

Tefkat es la definición e implementación de un lenguaje para transformación de los modelos. La propuesta fue realizada por de la Universidad de Queensland, Australia y se encuentra en una etapa de desarrollo bastante avanzada con respecto a otros lenguajes.

El lenguaje transforma modelos (M2M) en forma textual y ha adoptado un paradigma declarativo. Tefkat es totalmente compatible con los estándares de la OMG. Entiende que los modelos source y target definen dominios diferentes para cada uno de ellos, siendo ambos instancias de MOF.
Como desventaja, podemos mencionar que no hay declaración de pre ni de post condiciones.
Introduciéndonos en el cuerpo de la transformación, vemos que es posible declarar variables y metavariables, y que la separación sintáctica es provista por el lenguaje. De hecho, le aporta mucha legibilidad y claridad a la escritura de las reglas. Existen estructuras FORALL (algo de source) MAKE (algo del target) y FROM (algo de source) TO (algo del target) más la identación de las sentencias.
Tefkat también provee patrones (para el source) y templates (para el target) que se utilizan para nombrar restricciones que se pueden utilizar en más de una regla. Estas construcciones pueden a su vez parametrizarse, permitiendo la invocación a patrones y la recursión.

Las reglas se aplican en forma no determinística. Existe la aplicación condicional soportada por la cláusula IF-THEN-ELSE que permite la ejecución de reglas cuando la guarda es evaluada como verdadera. Las reglas pueden iterar y ejecutarse con recursión.

Para *traceability*, Tefkat incluye una cláusula LINKING que representa un *mapping* (asociación) entre los elementos del source y del target que participan de la transformación y que son almacenados una vez realizada la misma.

En este lenguaje es posible componer transformaciones mediante la definición de reglas, tomando transformaciones existentes y creando una nueva que preserva las relaciones.

UMLX

UMLX es un lenguaje gráfico de transformaciones entre modelos (M2M) y que se basó en extensiones mínimas a UML. Es una propuesta de E. Willink, del GMT Consortium. En un último avance, se ha anunciado que la transformación textual que la herramienta UMLX traduce desde el gráfico, puede también editarse. Como se traduce a lenguaje OCL, sería muy conveniente para aquellos que ya conocen este lenguaje y no se cae en la necesidad de conocer otro lenguaje.

La definición de este lenguaje se basó en QVT. Asimismo, los modelos participantes de las transformaciones, deben ser instancias de MOF. Cada uno de estos es tomado como un dominio diferente (uno de entrada o in y otro de salida o out), aunque se trate de los mismos modelos. Para estos no pueden especificarse pre o post condiciones.

En cuanto a las reglas de transformación, UMLX utiliza diferentes íconos gráficos para las transformaciones, para las reglas y para las relaciones de creación, preservación y eliminación de elementos. Cabe destacar que la semántica de los íconos del lenguaje no se ha definido, y aunque es gráficamente intuido, para algunas relaciones, no queda claro cuál es su significado.

Las estructuras que define este lenguaje son bastante simples e intuitivas, pero carecen de una semántica bien definida. No hay módulos, estructuras intermedias ni cláusulas de iteración o condición. Sólo existen variables locales, para indicar que se hace referencia a una instancia en particular. Por ejemplo, si se define una regla para todas las clases, y se define una regla también para cada atributo, al hacer referencia a una instancia de clase particular (cl, por ejemplo), se pueden mencionar a todos los atributos de cl, notándolo con @cl

La aplicación de las reglas se realiza de forma no determinística, por *pattern matching* de reglas y no se ha provisto ninguna construcción o claúsula que permita aplicar en forma condicional las reglas. Tampoco pueden invocarse otras reglas, ni se utilizan parámetros.

Ni la trazabilidad ni la composición han sido agregadas al lenguaje.

En las tablas 2 y 3 se presenta un resumen de las características enunciadas anteriormente y a modo de comparación directa entre los distintos lenguajes seleccionados.

Tabla 2. Comparación de lenguajes.

Característica/ lenguaje			**ATL**	**MOFScript**	**MOLA**
DEFINICIÓN	Tipo		Textual/ M2M	Textual/ M2Text	Gráfica/ M2M
	Estilo		Declarativo y operacional	Declarativo y operacional	Declarativo y operacional
	Pre y post condiciones		Si (OCL)	No	No
	MOF/QVT compatible		MOF: sí QVT: sí	MOF: sí QVT: sí	
REGLAS DE TRANSFORMACIÓN	DOMINIO	Lenguajes de dominio	Sí	Sí	Sí
		Dirección	Uni y bidireccional	Unidireccional	Unidireccional
		Relación entre origen y destino	Mismos o diferentes modelos	Diferentes modelos	Mismos o diferentes modelos
	CUERPO	Declaración de (meta)variables	Sí	Sí	Sí
		Patrones de transformaciones	No	No	Sí
		Separación sintáctica	Sí	Sí	No
		Estructuras intermedias	Sí	No	No
		Parametrización	Sí	Sí	Sí
	APLICACIÓN	Orden	No determinístico	Determinístico	Determinístico
		Aplicación condicional	No	Sí	Sí
		Iteración de reglas	Sí	Sí	Sí
	ORGANIZAC	Modularización	Sí	No	Sí
		Mecanismos de reuso	Sí	Sí	No
TRACEABILITY			Sí	Sí	Sí
COMPOSICIÓN			Sí	No	No

Tabla 3. Comparación de lenguajes (continuación).

Característica/ lenguaje			Tefkat	UMLX
DEFINICIÓN	Tipo		Textual/ M2M	Gráfico/ M2M
	Estilo		Declarativo	Declarativo y operacional
	Pre y post condiciones		No	No
	MOF/QVT compatible		MOF: sí QVT: sí	MOF: sí QVT: sí
REGLAS DE TRANSFORMACIÓN	DOMINIO	Lenguajes de dominio	Sí	Sí
		Dirección	Unidireccional	Unidireccional
		Relación entre origen y destino	Mismos o diferentes modelos	Mismos o diferentes modelos
	CUERPO	Declaración de (meta)variables	Sí	Sí
		Patrones de transformaciones	Sí	No
		Separación sintáctica	Sí	No
		Estructuras intermedias	No	No
		Parametrización	Sí	No
	APLICACIÓN	Orden	No determinístico	No determinístico
		Aplicación condicional	Sí	No
		Iteración de reglas	Sí	No
	ORGANIZAC	Modularización	Sí	No
		Mecanismos de reuso	Sí	No
TRACEABILITY			Sí	No
COMPOSICIÓN			Sí	No

4 CONCLUSIONES Y TRABAJO FUTURO

El desarrollo de software dirigido por modelos en una metodología que ha cobrado impulso desde su aparición en 2003. Tanto es así, que muchos trabajos se han propuesto y definido en cuanto a lenguajes, frameworks y herramientas que sirven de soporte y automatizan diferentes aspectos de MDA. Al adherirse a esta propuesta, es necesario elegir algunos de los lenguajes, lo cual no resulta una tarea sencilla. Nuestro trabajo ha presentado una extensión a un esquema de clasificación existente que nos permite evaluar características de los lenguajes de transformación de modelos. A su vez, realizamos una investigación sobre las diversas propuestas y seleccionamos algunas para evaluarlas en base a la clasificación antes mencionada.

De este análisis se desprende que tanto ATL como Tefkat son lenguajes muy completos y en avanzado desarrollo. Ambos adhieren a los estándares de la OMG, utilizan un lenguaje formal como es OCL y proveen mecanismos para traceability y composición de las reglas de transformación. ATL hace uso además de estructuras intermedias que mejoran la legibilidad y el reuso. Dentro de los lenguajes gráficos, MOLA es una buena propuesta: con buena legibilidad, intuitiva, fácil de

comprender y de utilizar.
Como líneas de trabajo futuro, podemos mencionar las siguientes: la mejora de ciertos aspectos a la propuesta de lenguaje mínimo para transformaciones de modelo que el grupo de investigación está desarrollando, en base a los lenguajes analizados; y la incorporación a esta misma propuesta de mecanismos que soporten traceability. Ambas líneas, se integrarían a la herramienta CASE [22] que nuestro grupo de investigación está desarrollando.

REFERENCIAS

[1] MDA Guide, v1. 0. 1, omg/03-06-01, June 2003. http://www. omg. org.
[2] OMG (Object Management Group) http://www. omg. org
[3] MOF 2. 0 Query/View/Transformations (QVT) - OMG Adopted Specification. March 2005. http://www. omg. org.
[4] Meta Object Facility (MOF) 2. 0. OMG Adopted Specification. 2003. http://www. omg. org.
[5] OMG. The Object Constraint Language Specification – Version 2. 0, for UML 2. 0, revised by the OMG, http://www.omg.org, April 2004.
[6] Jouault F. , Kurtev I. Transforming Models with ATL Workshop in Model Transformation in Practice at the MoDELS 2005 Conference. Montego Bay, Jamaica, Oct 3, 2005
[7] Marschall, F., Braun, P.: BOTL - The Bidirectional Object Oriented Transformation Language. Instituto de Informática, Universidad Técnica de Munich. Munich (2003)
[8] Agrawal, A., Kalmar, Z., Karsai, G., Shi, F., Vizhanyo, A.: GReAT User Manual.Nashville: Institute for Software-Integrated Systems, Vanderbilt University (2003)
[9] Sun Developer Network: Java Metadata Interface (JMI). SUN (2002)
http://java.sun.com/products/jmi/
[10] Akehurst, D.H., Howells, W.G., McDonald-Maier K.D.: Kent Model Transformation Language. En: MoDELS 2005 Conference. Montego Bay, Jamaica (2005)
[11] M. Peltier, J. Bézivin, and G. Guillaume. MTRANS: A general framework based on XSLT for model transformations. In WTUML'01, Proceedings of the Workshop on Transformations in UML, Genova, Italy, April 2001
[12] Model transformation- Inria. Mod-Transf ('04). http://modelware.inria.fr/rubrique15.htlm
[13] Eclipse org & Modelware. MOFScript (2005). http://www.eclipse.org/gmt/mofscript/
[14] Kalnins A., Barzdins J., Celms E. Model Transformation Language MOLA. Proceedings of MDAFA 2004, University of Linkoping, Sweden, 2004, pp.14-28.
[15] Tratt, L. The MT model transformation language. In MT 2006, Proceedings of the 2006 ACM symposium on Applied computing, pages 1296 - 1303
[16] Akehurst D , Howells W. , McDonald-Maier K. Model Transformation Language. Workshop in Model Transformation in Practice - MoDELS 2005 Conference, Jamaica, Oct 3, 2005
[17] Program-Transformation.Org. Stratego: Strategies for Program Transformation. Program-Transformation (2004). http://www.strategolanguage.org/Stratego/WebHome
[18] Lawley M. , Steel J. Practical Declarative Model Transformation with TefKat. Workshop in Model Transformation in Practice - MoDELS 2005 Conference. Jamaica, Oct 3, 2005
[19] Willink, E. UMLX - A graphical transformation language for MDA. En: OOPSLA 2003 Conference. Anaheim, California (2003)
[20] eXecutable UML.OMG proposal (2005). http://www.omg.org/cgi-bin/doc?ad/2005-4-2
[21] Czarnecki, Helsen. Feature-based survey of model transformation approaches.IBM System Journal,V45,N3, 2006
[22] Pons C., Giandini R., Pérez G., et al. Precise Assistant for the Modeling Process in an Environment with Refinement Orientation. "UML Modeling Languages and Applications: UML 2004 Satellite Activities, Revised Selected Papers". LNCS #3297. Springer, 2004.